AFRL-IF-RS-TR-2005-24
In-House Report
January 2005

THE DESIGN OF A FREQUENCY DOMAIN INTERFERENCE EXCISION PROCESSOR USING FIELD PROGRAMMABLE GATE ARRAYS

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

AFRL-IF-RS-TR-2005-24 has been reviewed and is approved for publication

APPROVED: /s/

STEPHEN C. TYLER
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 074-0188*

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE JANUARY 2005	3. REPORT TYPE AND DATES COVERED In-House Final, Oct 94 – Jun 04

4. TITLE AND SUBTITLE THE DESIGN OF A FREQUENCY DOMAIN INTERFERENCE EXCISION PROCESSOR USING FIELD PROGRAMMABLE GATE ARRAYS	5. FUNDING NUMBERS C - N/A PE - 602702F PR - 4519 TA - 42 WU - 89
6. AUTHOR(S) Stephen C. Tyler	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGC 525 Brooks Road Rome New York 13441-4505	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFGC 525 Brooks Road Rome New York 13441-4505	10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2005-24

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Stephen C. Tyler/IFGC/(315) 330-3466/ Stephen.Tyler@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

This report describes in-house work performed by AFRL which had a two-fold objective: development of a software/hardware testbed, and the design of a communications sub-system using this testbed. The report discusses the methodology involved in developing a testbed which utilized Field Programmable Gate Array (FPGA) technology. Several methods of programming FPGAs is discussed (writing Hardware Description Language vs. using schematic-based tools). An interference mitigation filter was subsequently developed using the testbed. The filter utilizes the frequency domain in order to identify and remove stationary and non-stationary narrowband interference from a direct sequence spread spectrum waveform.

14. SUBJECT TERMS Field Programmable Gate Array, Communications, Direct Sequence Spread Spectrum, Interference Mitigation, Frequency Domain	15. NUMBER OF PAGES 27
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1.0 Introduction

This report will discuss research and development work performed at the Air Force Research Laboratory's Rome Research Site (AFRL-RRS). Government personnel from the Information Connectivity Branch (IFGC) performed this work in-house. The Information Connectivity Branch is concerned with, in part, wireless applications as they relate to Air Force systems.

In order to test and evaluate advanced communications and signal processing concepts, IFGC has developed several prototyping facilities, or 'testbeds'. Collectively, these testbeds utilize a wide range of today's advanced digital hardware, including: multiple digital signal processors (DSPs), application specific integrated circuits (ASICs), and field programmable gate arrays (FPGAs). As one would expect, each technology has strengths and weaknesses; thus researchers match the algorithm with the most appropriate device technology.

This report will focus on one of these rapid prototyping technologies, namely, FPGAs [1, 2]. The FPGA was introduced as a reprogrammable logic device for use in any situation in which the logic configuration of the device needed to be changed periodically. For example, prototyping digital designs became popular using FPGAs, or creating products that could be 'field upgraded' at some future point-in-time as algorithms improved or standards changed. FPGAs debuted in the mid-1980's with several thousand programmable logic gates; today's state-of-the-art devices can provide over ten million. As these devices became more sophisticated, they added a new role: an alternative to Application Specific Integrated Circuits when production volumes were low.

1.1 The Field Programmable Gate Array

In a simplistic sense, an FPGA is a digital device consisting of 3 major components. IOBs, or input/output blocks, allow signals to enter and exit the device via the external 'pins' of the chip. Today's devices can have hundreds of pins, allowing massive amounts of data movement into and out of the chip. Next, a matrix of user programmable "logic cells" (referred to as combinational logic blocks or CLBs by one major FPGA vendor), are configured in order to implement the engineers design. These logic cells vary considerably from vendor to vendor, and even within the vendors' product line. And finally, routing resources are used to interconnect logic cells and to bring signals to and from IOBs.

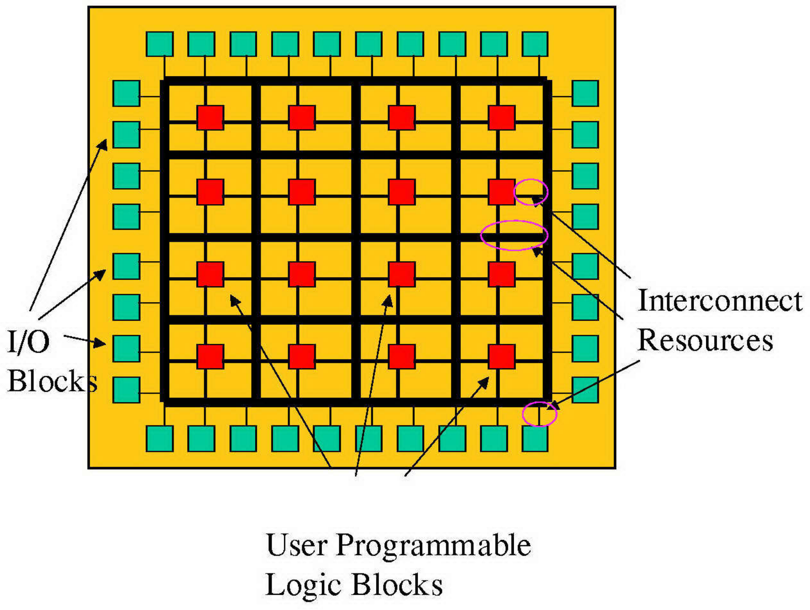


Figure 1.1-1: A Simple Representation of a Basic FPGA. I/O blocks route signals into and off of the chip. Logic blocks implement the user's design. Interconnect resources tie IOBs and Logic Blocks together.

FPGAs have become much more sophisticated now with on-board memory, embedded multipliers, and even imbedded fixed-point processors. The reader is encouraged to view the manufacturer's links provided for the latest on this quickly evolving technology [1, 2].

1.2 Design Methodology

FPGA synthesis tools use a Hardware Description Language (HDL), such as VHDL or Verilog, in order to produce the 'logic' that will be used by the FPGA to implement one's design. A very important question arises: how does the designer produce this 'HDL code' that describes the design? To answer this question was, in fact, one of the goals of this research effort. There are various design paths currently being used. Two of the most common forms of design entry will be discussed, namely, writing HDL code and schematic-based design.

Some designers prefer writing a "program like" HDL file which represents the logic within the design. This is a powerful, compact, and precise way to approach the problem. Each design component has a file associated with it, and once written, can be used repeatedly throughout the design. Designing new components often does not require 'starting from scratch', as one can use preexisting files for a launching point. The following is an example of an HDL code. In this instance, the language is VHDL and the code represents a two input OR gate:

```
library ieee;
use ieee.std_logic_1164.all;

entity OR_ent is
port(       x: in std_logic;
            y: in std_logic;
            F: out std_logic
);
end OR_ent;

architecture OR_arch of OR_ent is
begin

  process(x, y)
  begin
    -- compare to truth table
    if ((x='0') and (y='0')) then
          F <= '0';
        else
          F <= '1';
        end if;
  end process;

end OR_arch;

architecture OR_beh of OR_ent is
begin

  F <= x or y;

end OR_beh;
```

Another common approach is to use some form of schematic entry tool. The user creates a design using components from a preexisting library within a graphical user interface. Virtual wires are used to interconnect components together or connect components to IOBs. In this approach, the symbols represent HDL files; once the design is complete, vendor software is responsible for integrating these files into an overall HDL file representing the design. Figure 2 displays the graphical user interface (GUI) of the Xilinx Engineering Capture System (ECS), part of the Xilinx Integrated Software Environment (ISE). In typical schematic form, components are connected using lines, which represent single circuits, or thicker lines, which represent buses. Components have inputs on the left and outputs on the right. Special symbols, referred to as I/O markers, denote inputs and outputs to the FPGA (via the IOBs). The important thing to note here is that this schematic may look like a circuit board layout, with several stand-alone chips being interconnected by bus and circuit traces. This is not the case; the whole design will be 'fit' into a single FPGA.

The Xilinx ECS software tool is obviously only going to work for Xilinx FPGAs. Thus each FPGA vendor sells specific tools for their specific product line – all highly proprietary. There are other vendors who sell tools that, through various methods, produce 'generic' HDL code. This code can then be used for various FPGA devices; the device vendor's software will take that generic code and convert it for use with their particular architecture.   One such generic tool is CoWare's Signal Processing Worksystem [3] or SPW. SPW is first and foremost, a digital signal processing system-level simulator. SPW is typically used first to verify a concept from a high-level standpoint. But, it can also be used to test and simulate fixed-point designs and create HDL files from these designs. SPW was used in both roles for this in-house effort. Figure 1.2-2 shows the relationship to the Xilinx development system.

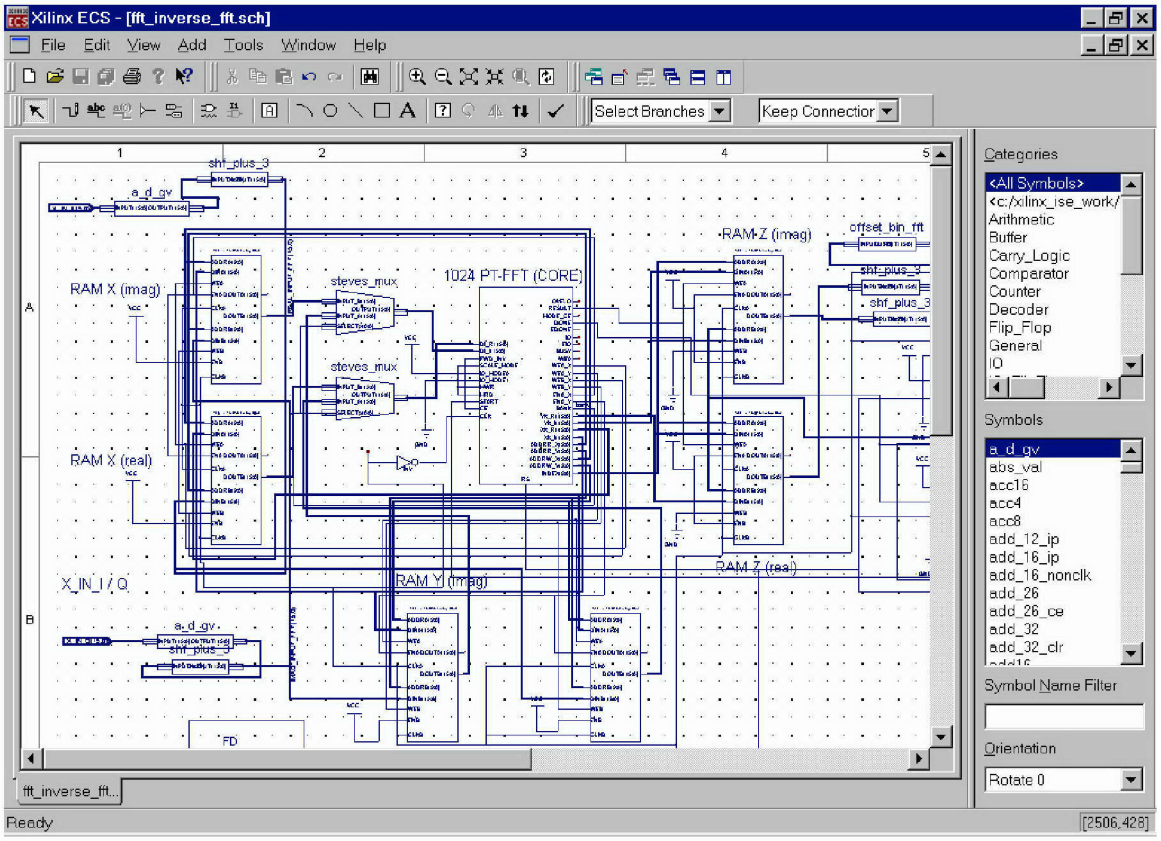


Figure 1.2-1: Xilinx Engineering Capture System. This software allows the designer to create schematic-like diagrams representing the design that will be implemented on the FPGA.

In addition to the above approaches, design components can be supplemented with Intellectual Property (IP) cores. IP cores are ready-made functions that range in complexity, but are typically components which are much more sophisticated than standard library components, and may be highly proprietary. Fast Fourier Transforms (FFTs) and Digital Down Converters (DDCs) are two examples of IP cores. Vendor software included with the IP cores typically allow for parameter adjustment within the specific components, for example, adjusting the size of the FFT, or coefficients within a filter. IP cores can be supplied from a large number of vendors. The cost of IP cores varies greatly; the FPGA vendors sometimes give it away, but many times, for complicated cores, or proprietary cores, there is significant cost.

As a final outcome, all of these methods produce a HDL file which represents the design; a file the synthesis tool will use for synthesizing logic that will ultimately be 'placed' within the FPGA.

The question remained: design using 'schematic' or write HDL code? Obviously, both methods have advantages and disadvantages. One could argue that the approach of writing HDL code is more powerful, more compact, more portable, but also less intuitive, harder to debug, and harder for people not familiar with the design to understand. On-the-other-hand, one could argue that schematic-based design is more intuitive, has a smaller learning curve, and can be easier for others to understand, but limits you to libraries available (what if they don't have a component your design needs?), is less efficient, and is not 'portable' across FPGA vendors.

After much thought, a hybrid approach was chosen; schematic-based design would be used as much as possible and HDL (VHDL, in this case) generation used when needed. Actually, two forms of schematic-based design are used in this testbed, namely, SPW and ECS, as can be seen from Figure 1.2-2.

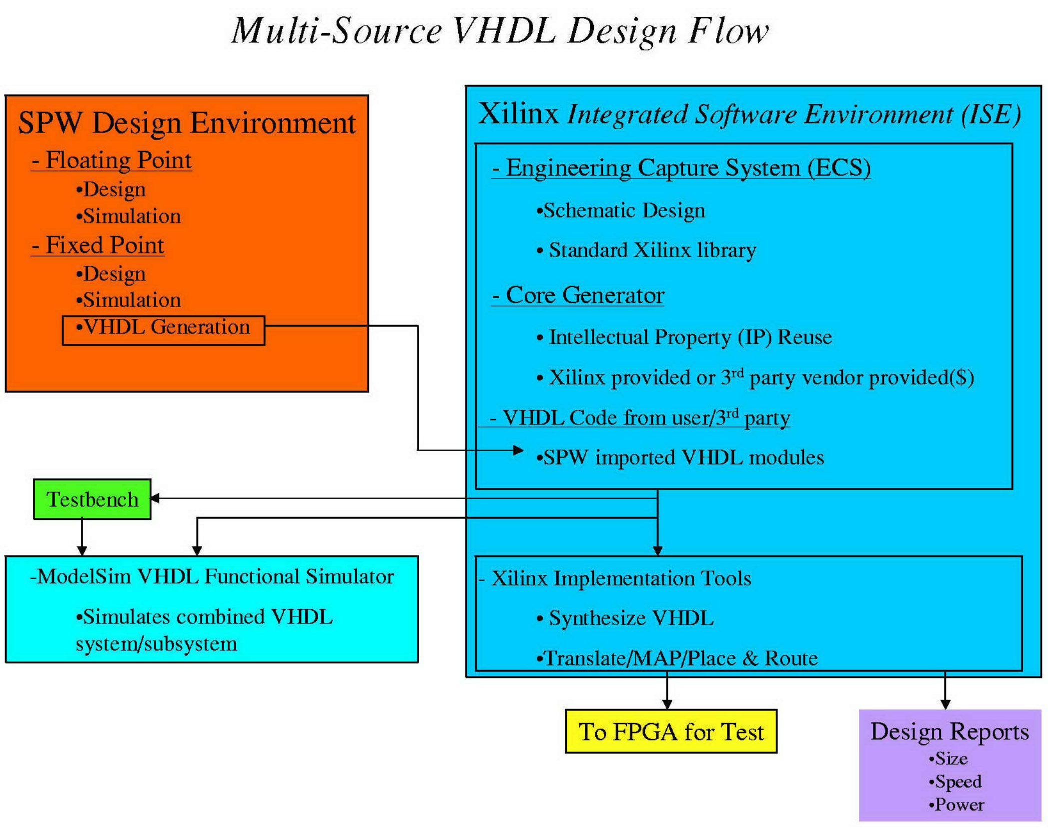

Multi-Source VHDL Design Flow
SPW Design Environment
- Floating Point
•Design
•Simulation
- Fixed Point
•Design
•Simulation
•VHDL Generation
Xilinx Integrated Software Environment (ISE)
- Engineering Capture System (ECS)
•Schematic Design
• Standard Xilinx library
- Core Generator
• Intellectual Property (IP) Reuse
• Xilinx provided or 3rd party vendor provided($)
- VHDL Code from user/3rd party
•SPW imported VHDL modules
Testbench
-ModelSim VHDL Functional Simulator
•Simulates combined VHDL system/subsystem
- Xilinx Implementation Tools
• Synthesize VHDL
•Translate/MAP/Place & Route
To FPGA for Test
Design Reports
•Size
•Speed
•Power

Figure 1.2-2: VHDL Design Flow. Various software packages allow multi-level simulation of design concepts in order to verify design and assist with debug. Xilinx Integrated Software Environment truly integrates many facets of design, including 3rd-party SPW developed VHDL.

2.0 Interference Excision Processor

2.1 Overview

Wireless communication systems, due to the fact that they transmit and receive signals over 'free space' channels, are susceptible to noise and interference. Military systems have the additional burden of operating in channels with hostile electromagnetic interference, i.e., a channel in which an adversary is intentionally trying to disrupt (aka 'jam') the communications signal. Many techniques have been developed over the years to make military communications signals more 'robust' to hostile channels. Some techniques, such a Direct Sequence Spread Spectrum (DSSS), 'fortify' the transmitted signal in order to make it more resilient to jamming and/or interference. Other techniques attempt to remove channel impairments at the receiver. Frequency excision, which is the focus of this research, falls into the latter category. It is important to note that frequency excision is typically used in combination with a wideband waveform, such as a DSSS signal, for reasons which will become clear shortly.

The goal for this work was to implement, in real-time, a frequency domain-based interference excision processor using the FPGA techniques outline in the previous section. Frequency excision can be implemented at various stages in the demodulation process, but for ease of implementation and maximum portability between various wireless systems, implementation was performed at baseband.

This work was based in large part on the work of Mitre Corporation [4]. Their work resulted in an excision processor being developed specifically for the Global Positioning System (GPS). Their design was implemented on an ASIC device. Their journal article provided valuable insight into the concept of frequency excision. In addition, many implementation details were included, such as shortcuts for reducing logic resources.

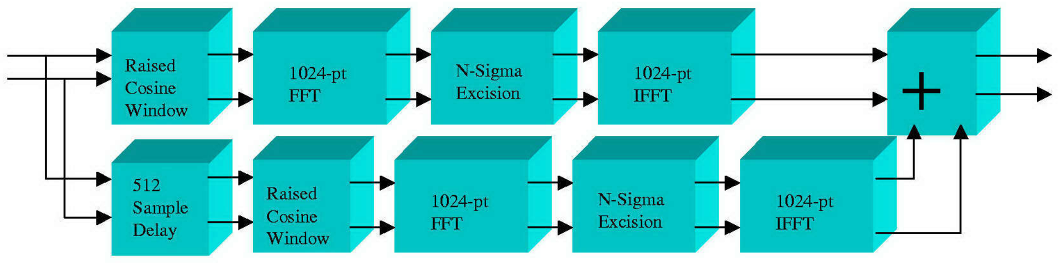


Figure 2.1-1: Conceptual Block Diagram. Forward and inverse FFTs surround excision algorithm. Windowing of received data samples necessitates 50% overlap, dual-path processing.

The conceptual block diagram for this concept can be seen in Figure 2.1-1. But before getting into any details, here is the simplified version: convert the received baseband signal (consisting of the DSSS signal, noise and jamming) to the frequency domain, look for and remove (hence the term excision) any frequency components which appear to be jamming, then convert the 'excised' signal back to the time domain for normal demodulation. This technique can therefore be thought of as a 'black box' process that can be used in a variety of communication systems.

It is important to note that when you remove frequency components which contain jamming, you are also removing the desired signal – the two are inseparable. Here is the reason for the DSSS signal; even with a portion of the spread spectrum waveform removed, the information signal can still be retrieved. Thus, when used together, DSSS and interference excision create a very robust 'system approach' against narrowband jamming.

Converting to the frequency domain and back again to the time domain is costly from a computational standpoint – so why is this done? The reason is to transform the received signal to an optimal 'domain' or 'space' in which to remove the jamming. Narrowband jammers (such as a continuous wave (CW) jammers) or partial band jammers (such as multiple CW, or CW jammers which have non-stationary frequencies, such as 'swept' or 'hopped' CW jammers) can be more easily identified and remove in the frequency domain rather than the time domain. Conversely, this technique would not work for jamming or interference which occupied the same bandwidth as the desired signal.

2.2 Implementation

Obviously, the key component in this design is the FFT and inverse FFT engine. The 'size' of the FFT refers to the number of data points used in the transform. As one may expect, there are many advantages to increasing the FFT size. For example, the more data points used, the greater the frequency resolution per output sample. In addition, larger transform sizes (along with windowing) make it easier to contain narrowband components. Disadvantages include more latency, more processing resources, more power, etc. A 1024-point transform was chosen for this design.

In order to quickly and efficiently produce a working excision system, it was determined early on that the task of building an FFT engine from scratch would not be feasible. Thus, an IP core would be the only solution. Several companies do offer FFT IP cores – for a fee. Xilinx offered a 'no cost' (included in the foundation tool set) FFT IP core engine which was complete with the exception of memory elements. The company included detailed instructions on how to build any one of three FFT systems using this core engine. The three systems offered were single, dual, and triple memory space configurations based on user requirements and available resources. The single memory space configuration forced the FFT core to use one memory element between input and output calculating cycles. The core, when configured like this, could not operate in a continuous fashion in terms of incoming samples. The double memory space configuration solved this problem, by having both an input and output memory space. And finally, the triple memory space configuration used a two input memory configuration in a 'ping-pong' arrangement to relax loading requirements of the double memory space design. The triple memory space configuration was chosen and successfully developed. This Xilinx FFT core offered several FFT block sizes, with a maximum size of 1024 samples (of real and imaginary data). For maximum resolution, the 1024-pt block size was chosen. A side note: because of the use of a 1024-pt FFT, data is grouped in 1K 'blocks' throughout this design.

Shortly after successfully developing the triple memory space (TMS) FFT, Xilinx offered a new and improved FFT IP core. The new FFT core handled all of the memory requirements internally, and, more importantly, utilized on-chip 18-bit by 18-bit multipliers found on the Xilinx Virtex-II FPGAs. The older TMS IP core was quickly replaced with the newer core. Thus, the TMS FFT core was replaced with a higher performing, less complicated (from a user development standpoint), less resource intensive, newer FFT core. This was especially significant due to the fact that 4 FFT processors were needed for this design. The joys of progress! Figure 2.2-1 is the top-level schematic, which was produced in the Xilinx ISE environment.

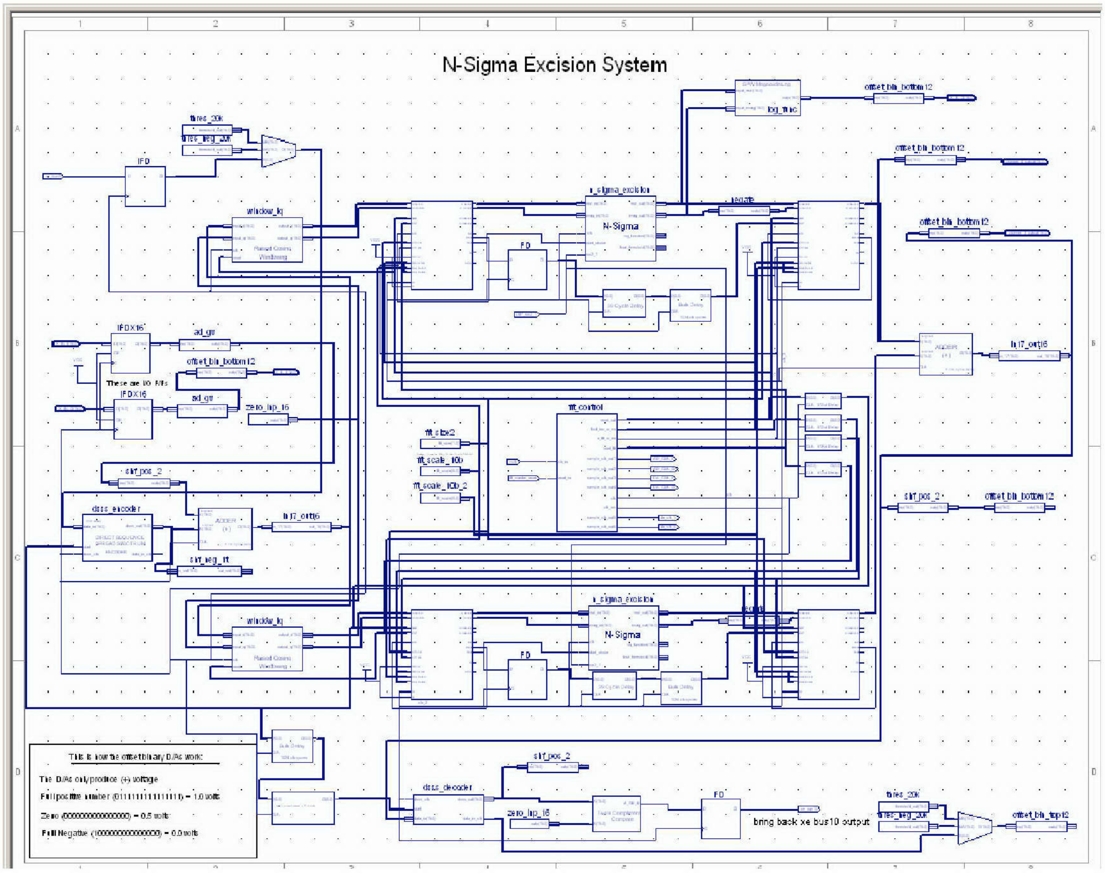
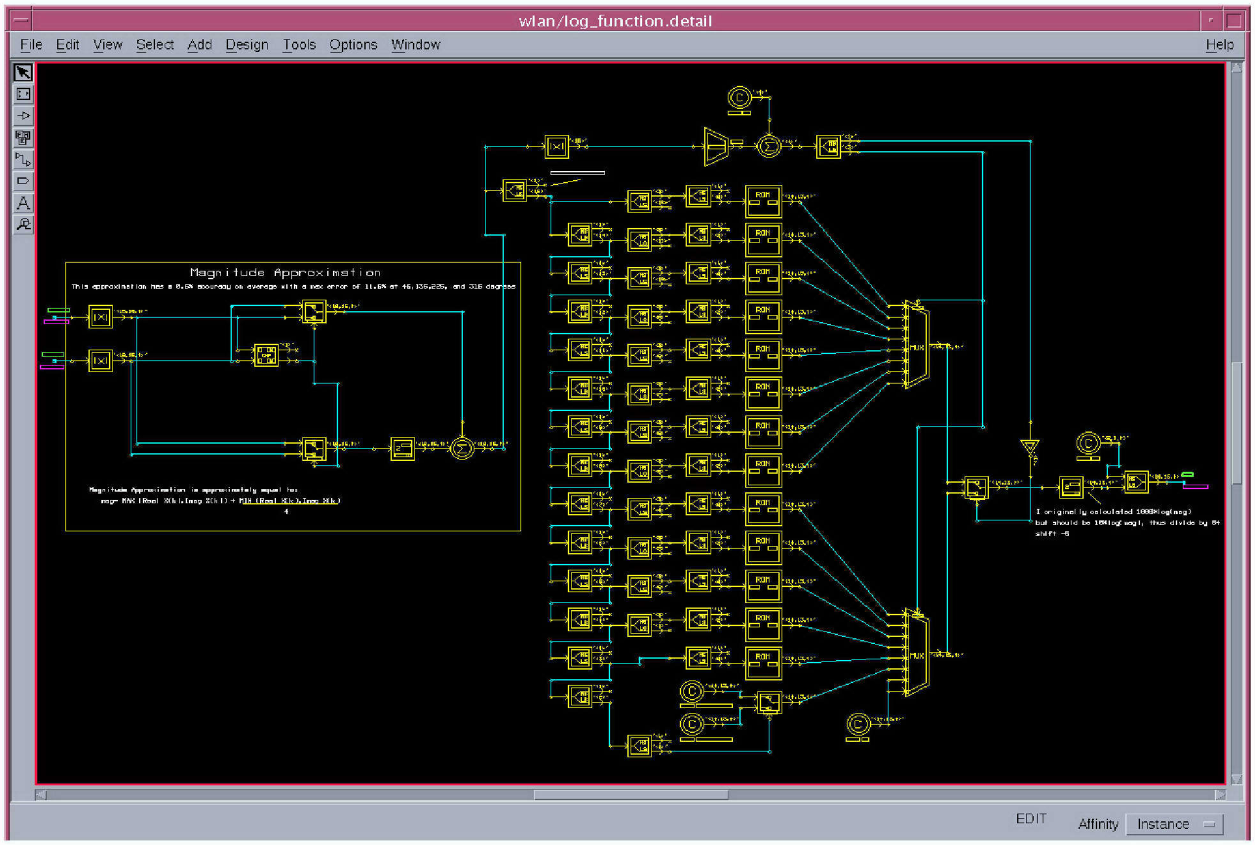

N-Sigma Excision System
n_sigma_excision
N-Sigma
fft_control
window_iq
dsss_encoder
dsss_decoder
IFD
IFDX16
FD
bring back xe bus10 output
This is how the offset binary D/As work:
The D/As only produce (+) voltage
Full positive number (0111111111111111) = 1.0 volts
Zero (0000000000000000) = 0.5 volts
Full Negative (1000000000000000) = 0.0 volts

Figure 2.2-1: N-Sigma top level schematic. This schematic contains not only the excision system, but also the DSSS transmitter and receiver. The analog jammer is combined with the transmitted signal at this level.

As can be seen in Figure 2.1-1, the first step in the excision process is data windowing. It is important to ‘contain’ the jamming signal in as few FFT bins as possible, because excision (or more simply stated, *removal*) of spectral components includes both jammer and desired signal. Therefore, the incoming data is windowed using a raised-cosine (amplitude weighting) windowing function in order to concentrate jammer energy in as few bins as possible. If the incoming samples are processed with no amplitude weighting function, jamming energy ‘leaks’ into the entire spectrum due to the spectral properties of the rectangular window, thus reducing the effectiveness of frequency excision. Windowing the incoming data has the effect of de-emphasizing samples at the beginning and end of a record – where discontinuities, and the corresponding spectral leakage, occur. An unfortunate result of windowing is the loss of information at the record boundaries. To overcome this deficiency, a dual processing path approach is taken, with 50% overlap of the incoming data. The two data streams are combined after processing. As a result, this requires twice the FFT, excision, and inverse FFT processing. As can be seen in Figure 2.1-1, the lower data path has a fixed 512 sample delay (1/2 of the 1024 block) as part of the 50% overlap processing.



Magnitude Approximation

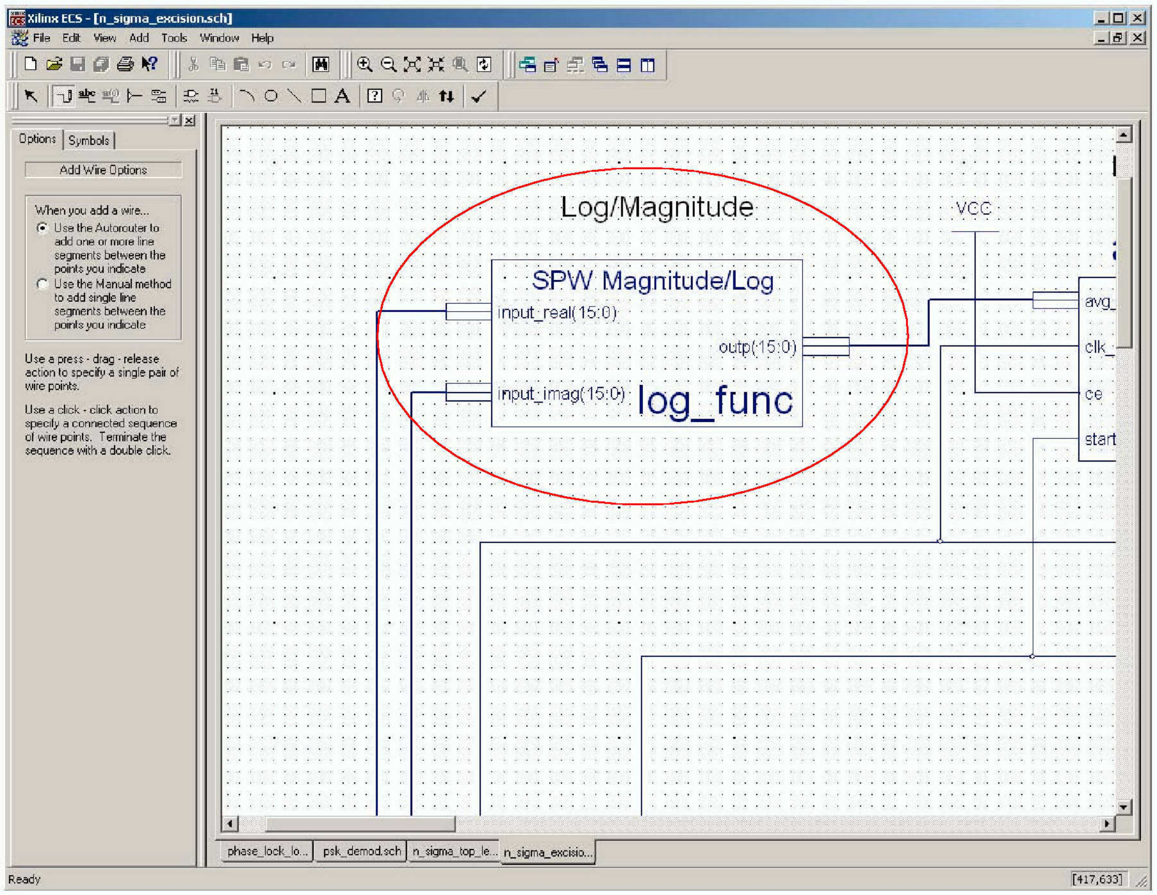
This approximation has a 0.6% accuracy on average with a max error of 11.6% at 45,135,225, and 315 degrees

Magnitude Approximation is approximately equal to:
mag= MAX(Real X(k),Imag X(k)) + MIN(Real X(k),Imag X(k))/4

I originally calculated 1000*log(mag)
but should be 10*log(mag), thus divide by 64
shift -6

Figure 2.2-2: Magnitude/Log Approximation Using SPW. A schematic-based design for the approximation of the magnitude and logarithm function was developed (and simulated) using SPW. The software then automatically created a VHDL file representing the block.

The next step is to perform the forward FFTs. The input and output samples used by the FFT core are 16-bit, with higher bit width utilized internally to the IP core. As stated earlier, a second generation Xilinx IP core was implemented which utilized the 18 by 18 bit on-chip multipliers. It is important to note that the output of the FFT is real and imaginary frequency samples; not the energy or power spectrum that is generally associated with 'frequency domain' graphs. All of the discussions so far have referred to the frequency *spectrum* which is the magnitude of the real and imaginary values produced by the FFT. Calculating the magnitude is the first operation performed by the excision block. A combined logarithm and magnitude function was developed using SPW, which can be seen in Figure 2.2-2. The concept was simulated within the SPW environment to ensure correct operation and to optimize the design. When the development was complete, the SPW software created a VHDL file. This file was imported into the Xilinx Integrated Software Environment (ISE) as a VHDL module. A symbol representing this imported code was automatically generated by ISE and added to a library of preexisting symbols for use within the Xilinx schematic editor. Figure 2.2-3 displays the SPW code after importing into the ISE environment.



Add Wire Options

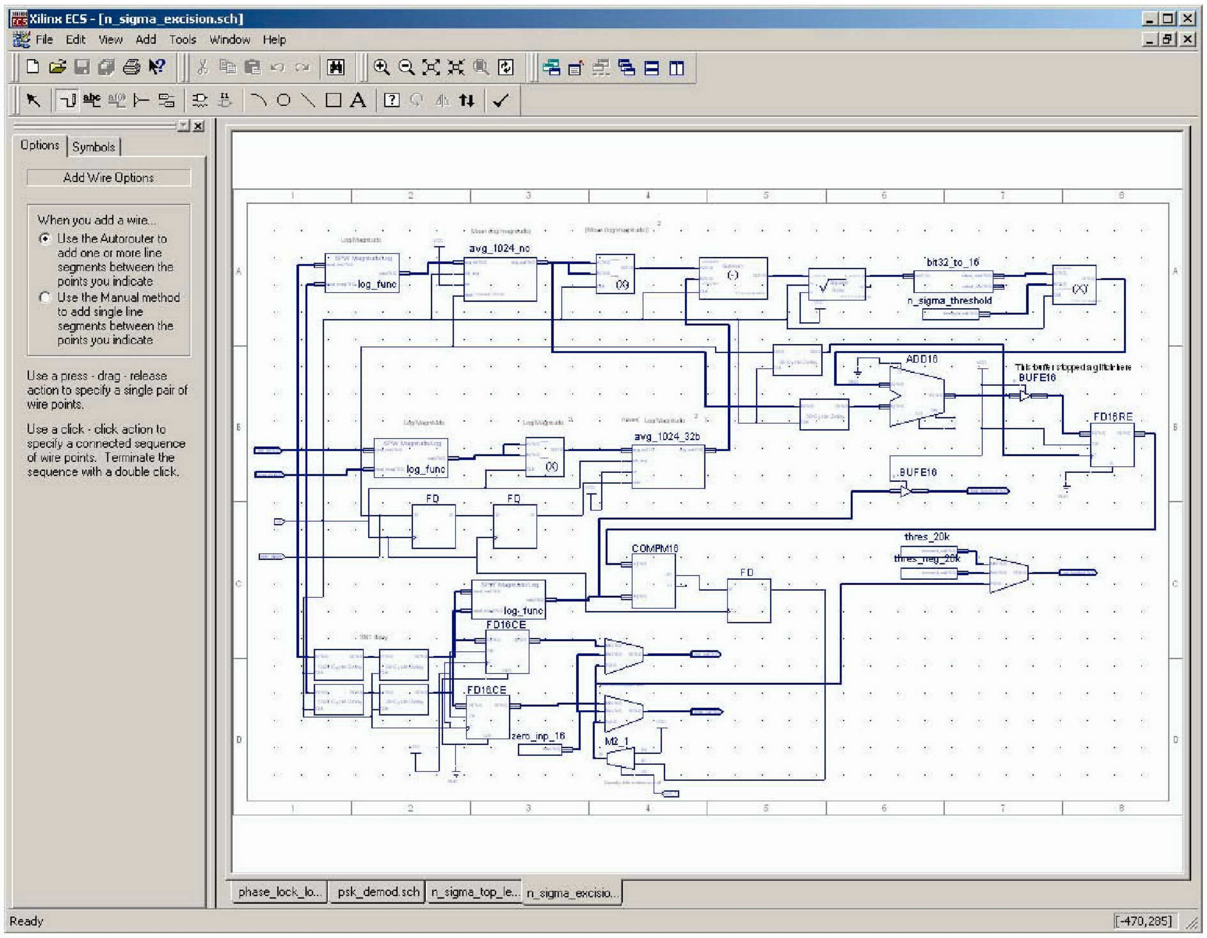
When you add a wire...
[✓ Use the Autorouter to add one or more line segments between the points you indicate]
Use the Manual method to add single line segments between the points you indicate

Use a press - drag - release action to specify a single pair of wire points.

Use a click - click action to specify a connected sequence of wire points. Terminate the sequence with a double click.

Figure 2.2-3: ECS Symbol of SPW Log Function. The magnitude/logarithm function of Figure 2.2-1 is shown now as a symbol in the Xilinx Integrated Software Environment. The SPW generated VHDL code was imported as a VHDL module and a symbol was generated representing this module.

The next stage is the excision algorithm itself. From a theoretical standpoint, the excision algorithm is not a particularly complicated one; it looks for frequency bins that are significantly 'outside' an average value. Since the desired signal is a DSSS waveform, the spectral characteristic are uniform and fairly flat; thus aiding in identifying jamming signals. Two key statistics that are calculated every 1K block: mean and standard deviation. The data output from the FFT is sequential; as a new complex sample arrives at the input, a processed sample is output (there is an initial delay before samples start to be output from the device, the delay value depends upon configuration of the FFT). The design of this overall system accommodates this sequential architecture. Therefore, in order to gather statistics about a block of samples, all samples need to be present. This necessitates a buffering of the frequency samples that are to be processed.



When you add a wire...

[✓ Use the Autorouter to add one or more line segments between the points you indicate]

Use the Manual method to add single line segments between the points you indicate

Use a press - drag - release action to specify a single pair of wire points.

Use a click - click action to specify a connected sequence of wire points. Terminate the sequence with a double click.

Figure 2.2-4: Excision Sub-system Schematic. Statistics of frequency samples (mean and standard deviation) are calculated here on a block by block basis in order to determine excision threshold value.

In order to obtain the standard deviation, the square root function needed to be implemented (remembering that the standard deviation is the square root of the variance). There are approximations for this function, due to its computationally intense nature. But after reviewing the Xilinx IP core offerings, it was decided to use their no-charge core.

Once the blocks statistics are gathered, the following formula is applied:

Excision Threshold = (mean + (scale factor)(standard deviation))*

Any samples above this threshold are simply set to zero. The scale factor was determined by experimentation and would vary depending upon several implementation issues.

After the excision unit, the frequency samples are then converted back to the time domain. It is important to note that in order to determine which frequency samples to excise, the magnitude function was used. All of the statistics were based upon magnitude samples, but ultimately, real and imaginary samples from the forward FFT were excised, not magnitude/phase samples. Thus, when converting back to the time domain, no magnitude conversion was necessary; these samples were simply input to the inverse FFT. After being processed by the inverse FFT, the real components were summed to create the ‘end result’ data stream. The next step, which is not considered part of the excision system, would be normal dispreading of the DSSS signal. A bypass function was incorporated which allows relative comparison of performance with and without the excision system present in the demodulation chain.

A timing sub-system was developed which included a Xilinx Digital Clock Manager (DCM) consisting a clock delay lock loop used to minimize clock skew. The timing sub-system also handled resets, FFT enable lines, D/A and A/D clocks, and system clocks.

3.0 Excision Test System

3.1 GV DSP Hardware Accelerator

In order to test the interference excision processor, a specialized FPGA board was purchased from a Xilinx ‘partner’ company, GV & Associates, Inc. The GVA-350 is referred to as a “Virtex-II hardware Accelerator”. This stand-alone board contains the following:

- 2 Virtex-II 6000s (XV26000) FPGAs for signal processing
- 2 Spartan-II FPGAs for external interface and configuration control
- 4 channels of 100 MSPS 12-bit analog-to-digital (A/D) conversion
- 4 channels of 100 MSPS 12-bit digital-to-analog (D/A) conversion
- 512K x 18 ZBT SRAM for each XV26000
- 4 16-bit Low Voltage Differential Signaling (LVDS) headers (34-pin)

- 25-pin parallel port
- USB interface
- 8Mx8 Flash EPROM
- On-board 50 MHz system clock
- 2 10-bit LEDS, 4-bit DIP switch

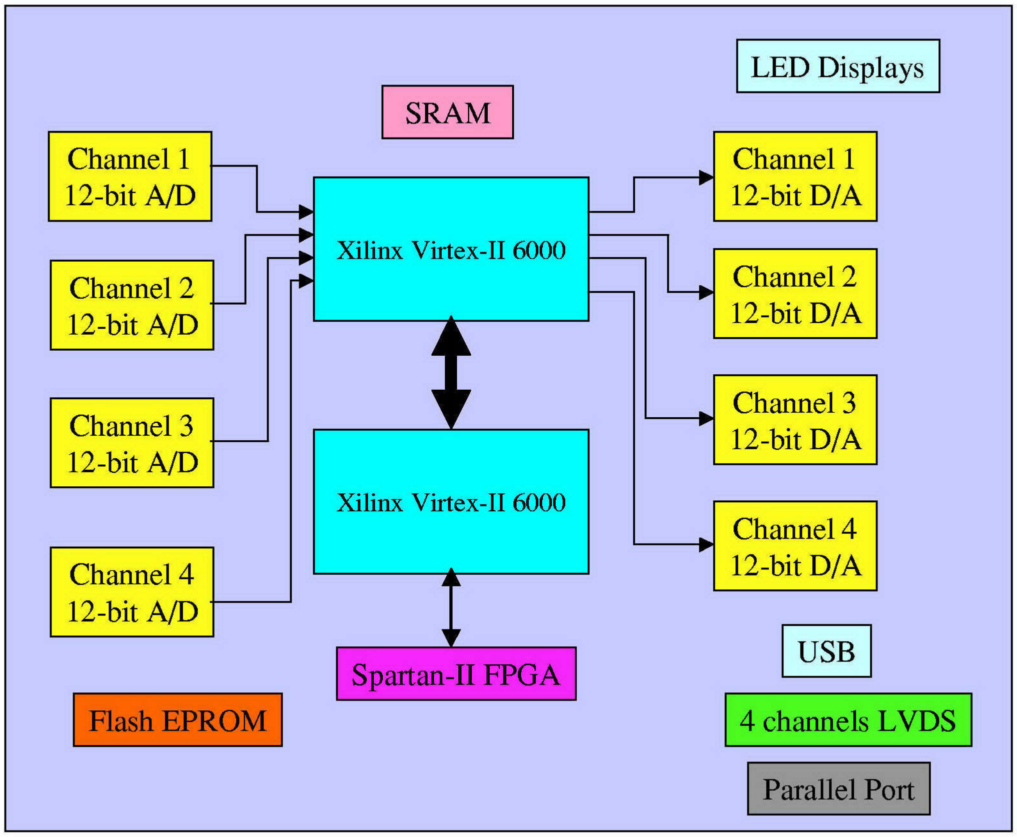


Figure 3.1-1: GVA FPGA Board. Simplified block diagram of the GVA-350 DSP hardware accelerator. Two Virtex-II 6000s and 4 channels of A/D and D/A are ideal for prototyping various DSP algorithms involving analog signals.

The GVA-350 is the ideal board for testing signal processing concepts which involve analog signals due to the fact that the board has 4 separate 100 MSPS, 12-bit channels of A/D and D/A. All eight channels are fed directly into/from one Virtex-II FPGA (referred to by the company as the 'analog' FPGA). A 203-bit bus connects this FPGA with the other Virtex-II (referred to as the 'digital' FPGA due to the fact that it interfaces with various ports such as USB, and LVDS).

3.2 Transmitter, Channel, and Receiver Configuration

In order to test the interference excision processor, a suitable Direct Sequence Spread Spectrum waveform had to be utilized. For simplicity, a DSSS waveform was developed within the FPGA. The transmitter can be seen in Figure 3.3-1 and the receiver in Figure 3.3-2. For the next phase of testing, the transmitter and channel will not be included within FPGA. To include as many 'real world effects' as possible for the jammer (A/D effects, for example), an analog signal source is used. A Hewlett Packard function generator was used as the jamming source. A Firebird 6000A bit error rate monitor was used to document performance. A bit pattern was sent from the Firebird, along with a clock signal, to the 25-pin 'parallel port' connector on the GVA-350. This signal was then fed to the DSSS transmitter for encoding. The transmitted DSSS signal was then added to channel 1 from the A/D converter. This is the 'simplified' channel. The result was either routed to the DSSS receiver (for baseline results) or routed to the excision filter followed by the DSSS receiver. Thus, relative performance gains were calculated rather than absolute gains. This configuration can be seen in Figure 3.2-1.

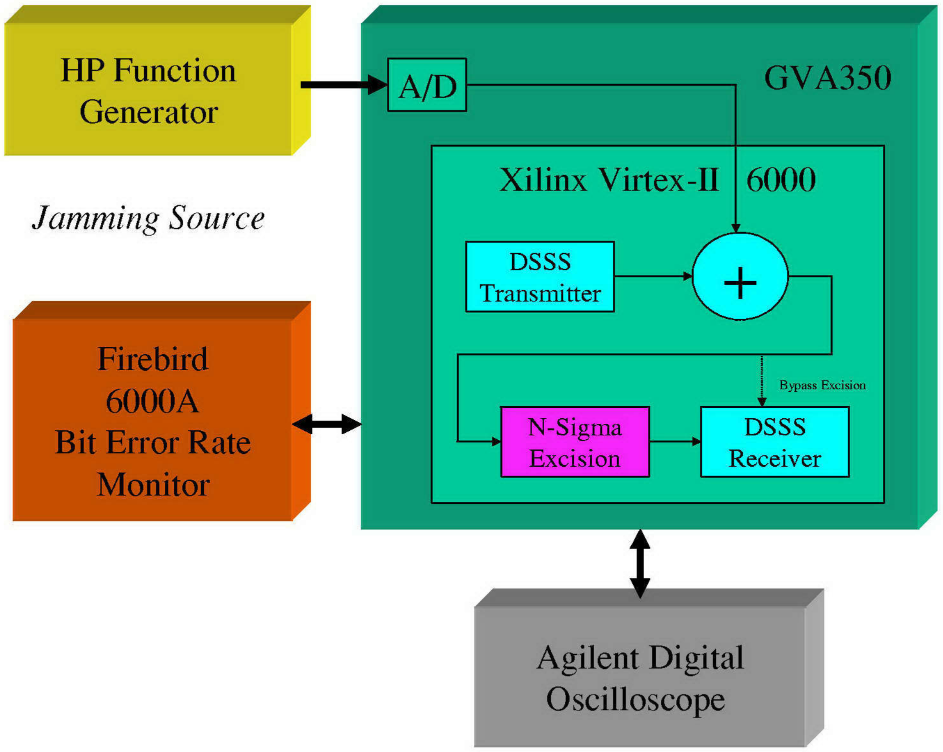


Figure 3.2-1: Simplified Block Diagram of Test System. Internally generated DSSS transmitter and receiver are tested with real analog jammer. Bit error rate monitor verifies performance.

3.3 DSSS Transmitter/Receiver

An 11-tap shift register was used to develop the DSSS waveform. The sampling rate for this design was set at 25 MHz. This sampling rate is also the internal clock rate of the processor. Every sample that is produced by the A/D converter gets processed by this highly pipelined excision processor. There is no buffering or decimation of incoming samples. With more time spent on optimizing the design from a logic perspective, and use of better FPGA place and route tools, a significantly higher clock rate could be achieved, but for this phase of the research, 25 MHz was sufficient.

Given the 25 MHz system clock and a requirement of 4 samples per pseudo-random (PN) DSSS chip, the PN chip rate equals 6.25 MHz. A processing gain of 128 was chosen, thus resulting in a data rate of 48.8 KHz. A series of counters were used to divide-down the system clock to achieve the PN chip rate and the data rate clocks. The data rate clock was fed back to the Firebird in order to generate the exact bit timing.

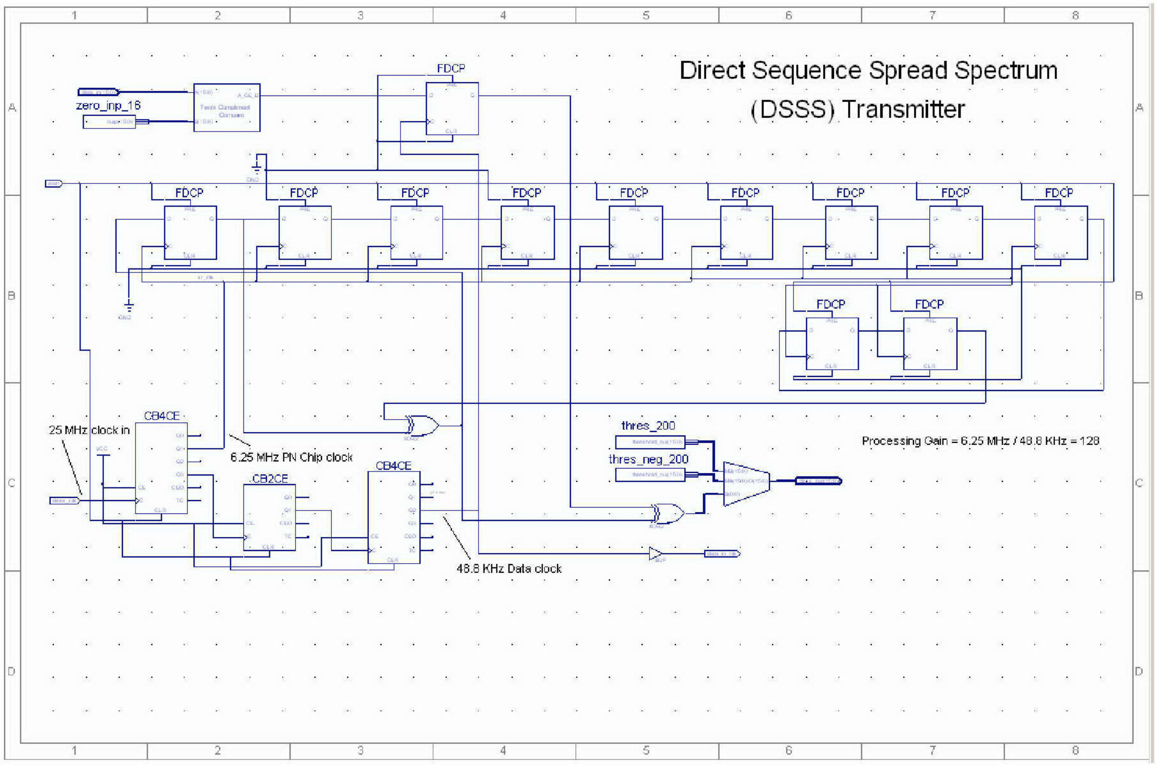


Figure 3.3-1: Direct Sequence Spread Spectrum (DSSS) Transmitter. This module (along with the receiver) was developed in order to test the excision system.

The details of the FFT processor will be discussed shortly, but it is important to note that it uses 16-bit input and output ports. Because of this, it was decided that all data bit widths would be set to this value. Therefore, compensation for the 12-bit A/D's and D/A's would be needed. For the A/D's, 'sign extension' or simply copying bits 13 through 16 with the most significant bit (MSB) of the A/D's 12 bits was all that was needed. For the D/A's, two conversions had to be made: first a reduction from 16 to 12 bits, and then a conversion to 'offset binary'. Offset binary is used by the D/A's because they are limited to an output voltage range of 0 to 0.5 volts (thus no negative voltage swings). Therefore, a full positive value for the D/A would be 0.5 volts and would represent the binary number '011111111111'. Zero (000000000000) would be 0.25 volts, and the largest negative number (10000000000) would equal 0.0 volts. SPW was used to create a converter for both the A/D's and the D/A's. It is important to keep in mind that it is more difficult to transition to the D/A than the A/D, due to the fact that 'scaling' of the 16-bit internal bus down to the 12 D/A is critical. Incorrectly discarding the upper or lower 4 bits can result in erroneous results.

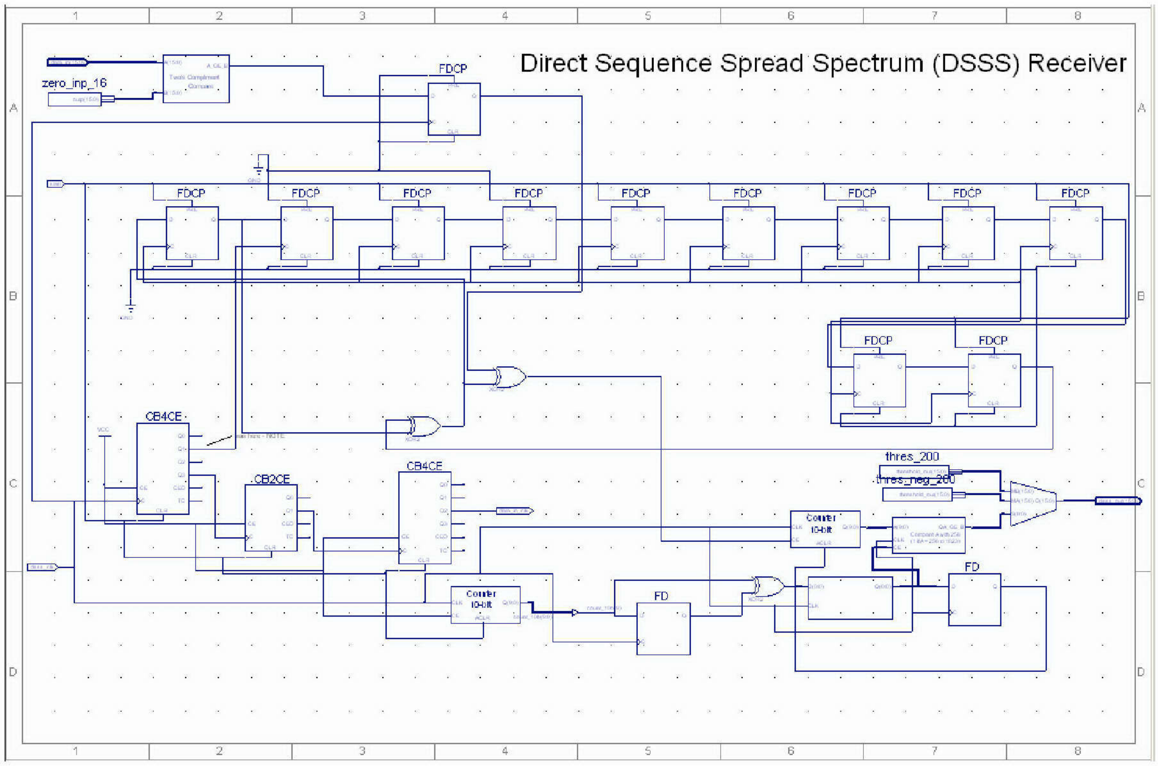


Figure 3.3-2: Direct Sequence Spread Spectrum (DSSS) Receiver. This module (along with the transmitter) was developed in order to test the excision system. Fixed channel delay, along with synchronous timing, result in ease of implementation.

The DSSS receiver was greatly simplified in this design due to the fact that synchronization was trivial. With the transmit and receive PN code timing error a constant delay, despreading was simply a matter of determining that delay. In addition, because both the transmitter and receiver use the same clock, clock recovery was not a issue. The diagram for the DSSS receiver can be seen in Figure 3.2-2. This schematic is very similar to the transmitter, with the exception of the 'integrate and dump' logic which is used for despreading of the DSSS signal.

4.0 Test Results

In order to demonstrate and verify correct operation of several subsystems in this overall design, various signals were routed to the 12-bit D/A converters of the GV board. Up to 4 signals could be analyzed at any one time. These signals were then sent to an Agilent Technologies 54622D Digital Oscilloscope. The following screen captures include descriptive summaries.

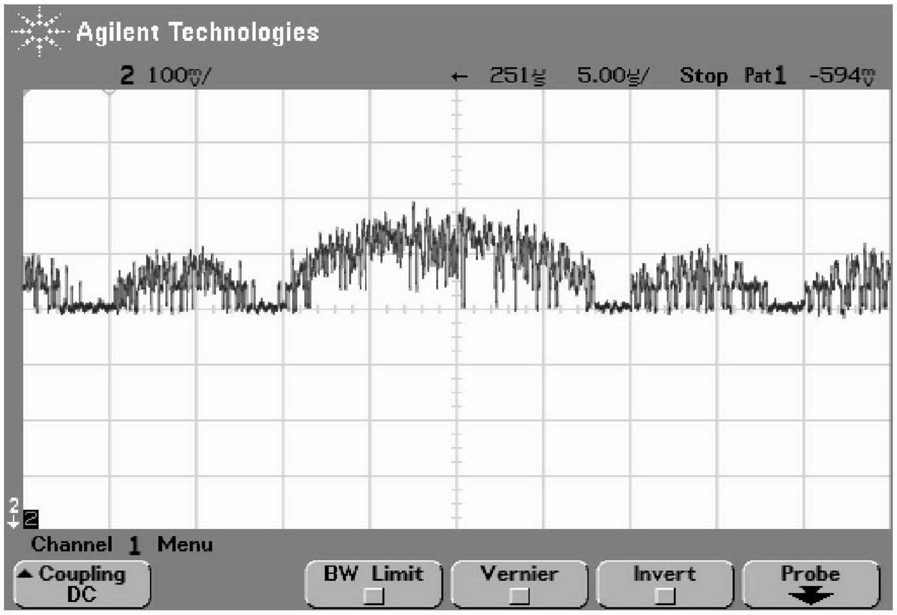


Figure 4.0-1: The DSSS Signal Magnitude Spectrum. This time-domain display (oscilloscope) actually represents frequency samples, i.e., the 16-bit output from the 1024-pt Fast Fourier Transform. This display represents the real and imaginary samples after they are sent to a log/magnitude translator.

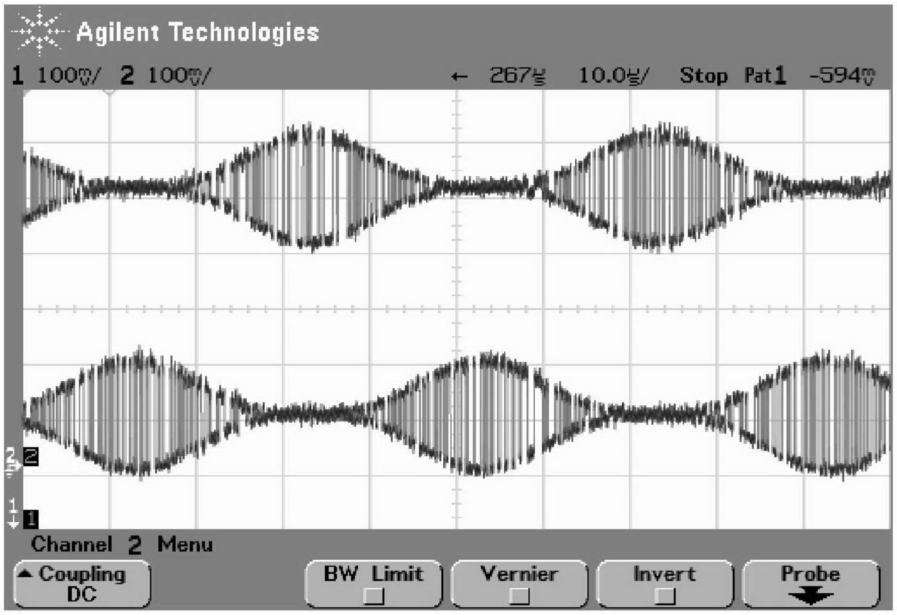


Figure 4.0-2: FFT-Processed Time Waveforms. The DSSS PN waveforms shown here after being processed by the frequency domain excision filter. Note the effect of staggered data paths and raised cosine windowing. In this example, no jammer was present, and thus, no frequency samples were excised.

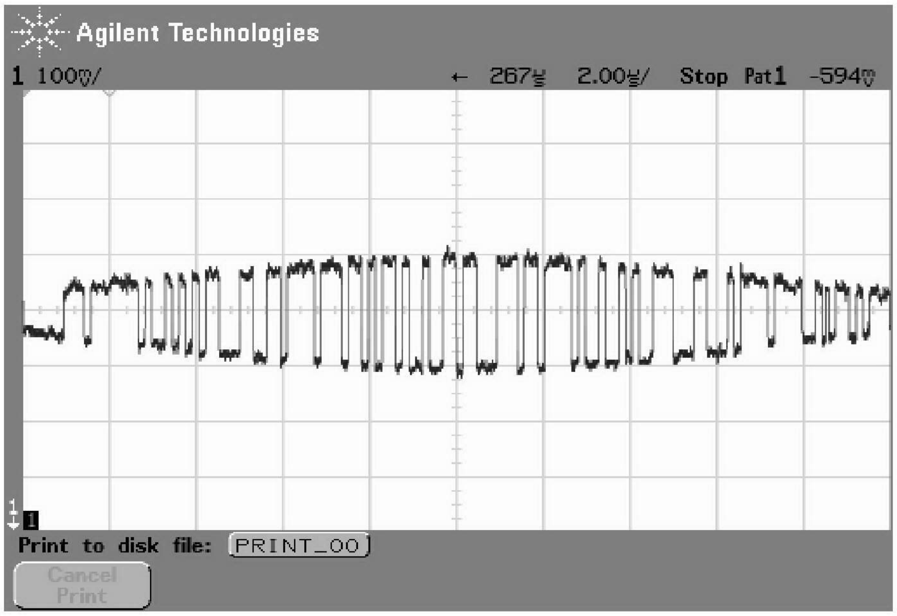

Agilent Technologies
1 100mV/
← 267µs 2.00µs/ Stop Pat1 -594mV
Print to disk file: PRINT_00
Cancel Print

Figure 4.0-3: Direct Sequence Spread Spectrum PN-chips. This close-up view of the in-phase channel of the DSSS waveform shows the effect of raised cosine windowing.

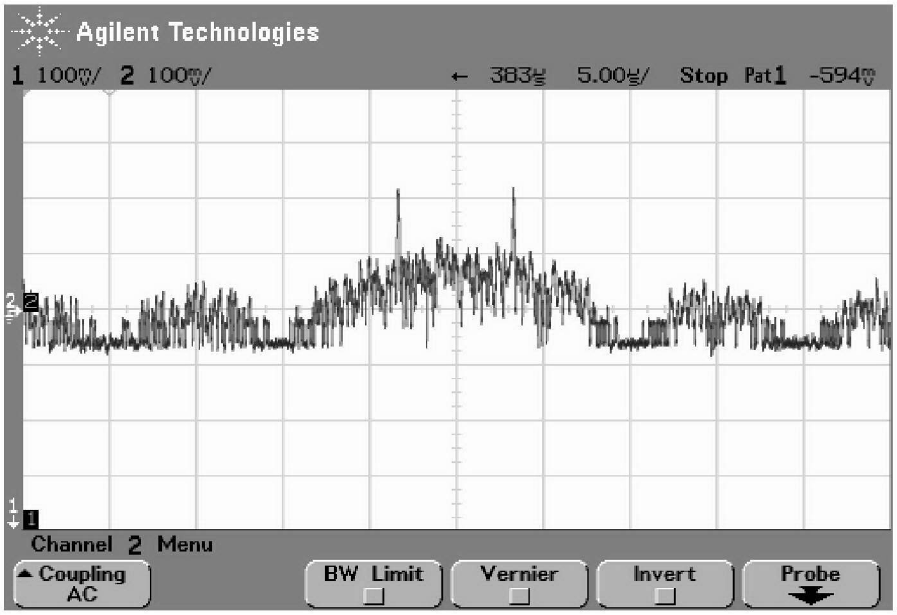


Figure 4.0-4: DSSS Signal Spectrum with Jamming. The DSSS signal (magnitude spectrum) with a CW jammer present. In this case, the excision filter is disabled. Note the spectral containment of the CW jammer due to the cosine windowing. Clearly, one can see the advantage of transforming to the frequency domain in order to precisely remove the jammer, and, at the same time, distorting the desired signal as little as possible.

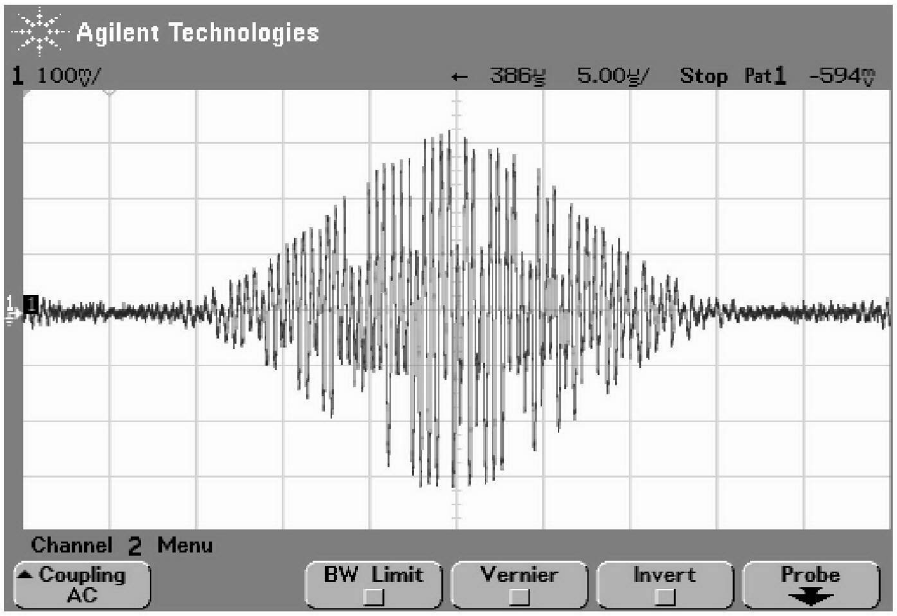


Figure 4.0-5: DSSS Signal with Jamming– Excision Disabled. This time domain view of the DSSS signal (in-phase only) demonstrates how the CW jammer distorts the waveform. In this case, the excision filter is disabled. Note the lack of any discernable DSSS PN-chips which were seen in Figure 4.0-3. One can clearly see that in this ‘domain’, excision is not possible.

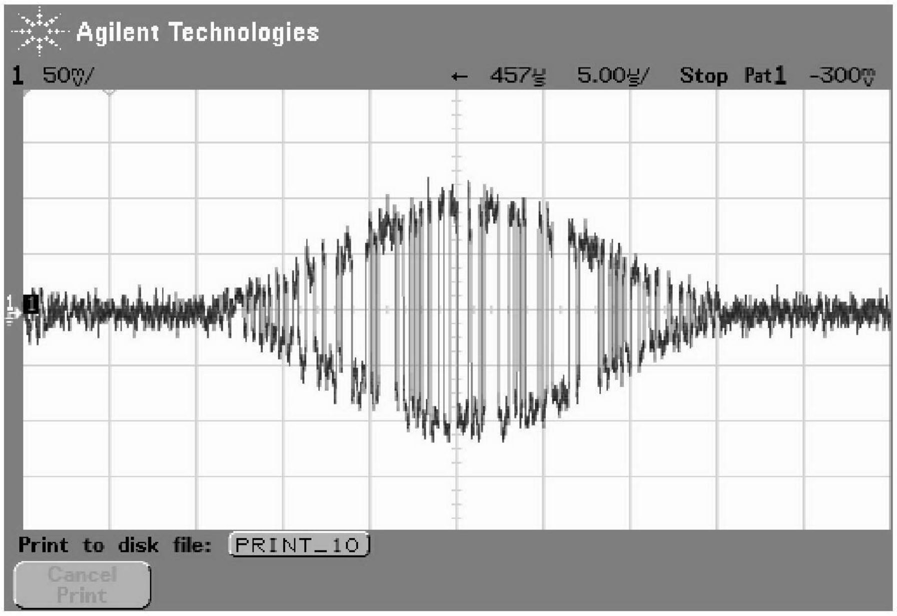

Agilent Technologies
1 50mV/
← 457µs 5.00µs/ Stop Pat1 -300mV
Print to disk file: PRINT_10
Cancel Print

Figure 4.0-6: DSSS Signal with Jamming – Excision Enabled. The DSSS signal (in-phase only) with the CW jammer present. In this case, the excision filter is enabled. Note the discernable DSSS PN-chips, which were absent in the previous figure.

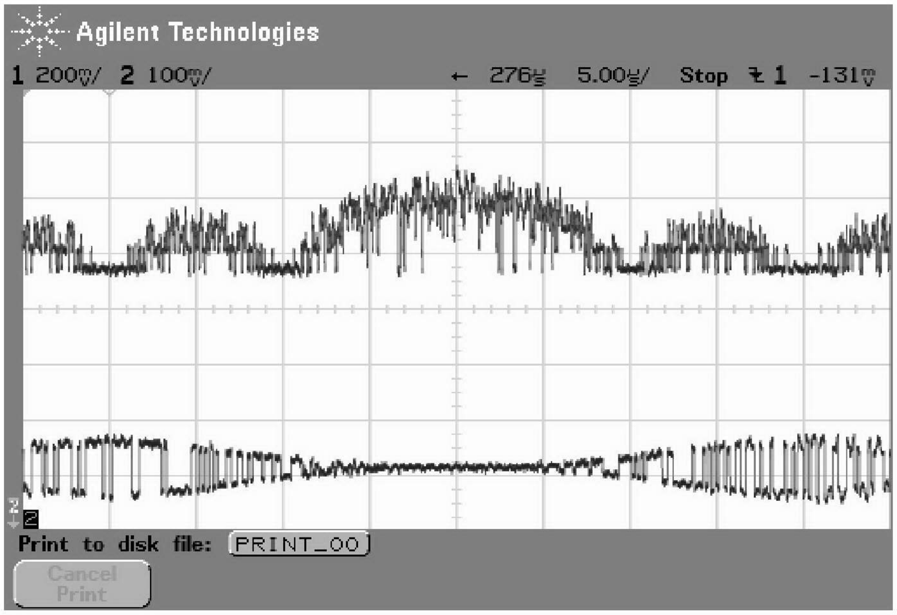


Figure 4.0-7: DSSS Signal Spectrum with Jamming – Excision Enabled. The DSSS signal (top: spectrum, bottom: in-phase time domain) with the CW jammer present. In this case, the excision filter is enabled. Note the absence of the CW spectral lines in the top waveform – which were present in Figure 4.0-4.

Probably the most important 'figure of merit' of this type of filter would be its jammer suppression. How much better can the desired spread spectrum signal do with the filter enable vs. disabled? Well, the short answer to that question is: that cannot be determined at this time due to the filter's robust performance. The performance envelope of the combination spread spectrum signal and excision unit is beyond that of the fixed-point numbering system and A/D converters of the test setup of Figure 3.2-1. In-other-words, the filter performs so well, we cannot test the full extent of its performance in its current test configuration.

Dynamic range:

16-bit fixed-point numbering system: 16 bit x 6 dB/bit = 96 dB

12-bit analog-to-digital converter: 12 bit x 6 dB/bit = 72 dB

The A/D converter has a 'realistic' dynamic range of approximately 68 dB due to implementation losses. Factoring in the DSSS signals 42 dB of processing gain, there is simply not enough dynamic range present to test the filter.

5.0 Conclusions

This effort had a two-fold objective: develop an efficient methodology for creating digital systems on FPGAs, and proving this methodology by successfully developing a complex digital system. The interference excision filter and DSSS system, with over 5 million logic gates, was successfully developed in a relatively short time - approximately 6 months. The Xilinx Engineering Capture System allows schematic level design of complex digital systems. The integration of CoWare's Signal Processing Worksystem with the Xilinx Integrated Software Environment (which includes the Engineering Capture System and the Xilinx LogiCore intellectual property suite) resulted in two software systems which greatly complemented each other, virtually eliminating the need to code in either VHDL or Verilog Hardware Description Language.

The interference excision filter will be used again in an upcoming R.F. watermarking demonstration system. The watermarking system includes a Quaternary Phase-Shift Keying (QPSK) receiver based upon the Costas loop carrier synchronizer [5]. This receiver has been successfully developed using this FPGA testbed. Thus, two complex digital systems have been successfully developed to date.

The schematic-based nature of this design methodology is very intuitive from an engineering standpoint, and can easily be understood by designers not familiar with this project. One could argue that revisiting the design, after a period of time, for re-use or re-design, is much easier and more intuitive than reviewing large amounts of VHDL code. After several years of use, we will see if this argument holds true.

Future plans for this testbed include integration with the DSP-based in-house testbed and upgrades to the FPGA hardware, thus allowing maximum flexibility and performance when designing complex Air Force demonstration systems.

6.0 References

[1] http://xilinx.com/

[2] http://altera.com/

[3] http://www.coware.com/

[4] P. T. Capozza, B. J. Holland, T. M. Hopkinson, and R. L. Landrau, "A Single-Chip Narrow-Band Frequency-Domain Excisor for a Global Positioning System (GPS) Receiver," in IEEE Journal of Solid-State Circuits, VOL. 35, NO. 3, Mar 2000.

[5] S. Benedetto, E. Biglieri, and V. Casellani, "Digital Transmission Theory". Englewood Cliffs, N.J.: Prentice-Hall.